

SPECULATIVE MULTI-THREAD PROCESSING METHOD AND ITS DEVICE

Patent Number: JP2000047887
Publication date: 2000-02-18
Inventor(s): OTSU KANEMITSU; BABA KEISHIN; KOYANAGI SHIGERU
Applicant(s): TOSHIBA CORP
Requested Patent: ☐ JP2000047887 (JP00047887)
Application Number: JP19980215945 19980730
Priority Number(s):
IPC Classification: G06F9/46 ; G06F15/16
EC Classification:
Equivalents:

Abstract

PROBLEM TO BE SOLVED: To provide a speculative multi-thread processor capable of improving the processing speed of a program.

SOLUTION: This speculative multi-thread processor is provided with plural processor parts PE0 to PEn-1 respectively executing an assigned thread, a thread manager 1 integrally managing the execution of a thread by controlling each part PE0 to PEn-1 and a memory part 2 storing a processing result, etc., by each part PE0 to PEn-1. The manager 1 assigns respectively different threads to each processor part to simultaneously and speculatively execute plural program path to improve the processing speed of the program. In addition, the manager 1 transfers data by using a register or a load store buffer in each processor part in the case of generating the necessity of transferring data between threads.

Data supplied from the esp@cenet database - I2

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2000-47887

(P2000-47887A)

(43) 公開日 平成12年2月18日 (2000.2.18)

(51) Int.Cl. ⁷	識別記号	F I	テマコード (参考)
G 0 6 F 9/46	3 6 0	G 0 6 F 9/46	3 6 0 B 5 B 0 4 5
15/16	4 3 0	15/16	4 3 0 Z 5 B 0 9 8

審査請求 未請求 請求項の数11 O L (全 16 頁)

(21) 出願番号 特願平10-215945

(22) 出願日 平成10年7月30日 (1998.7.30)

(71) 出願人 000003078

株式会社東芝

神奈川県川崎市幸区堀川町72番地

(72) 発明者 大 津 金 光

栃木県宇都宮市峰1-19-14 あすなろハイツ202

(72) 発明者 馬 場 敬 信

栃木県宇都宮市西の宮町1176-103

(72) 発明者 小 柳 滋

神奈川県川崎市幸区小向東芝町1 株式会社東芝研究開発センター内

(74) 代理人 100064285

弁理士 佐藤 一雄 (外3名)

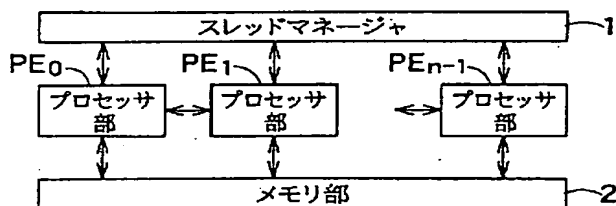
最終頁に続く

(54) 【発明の名称】 投機的マルチスレッド処理方法および投機的マルチスレッド処理装置

(57) 【要約】

【課題】 プログラムの処理速度を向上できる投機的マルチスレッド処理装置の提供。

【解決手段】 本発明の投機的マルチスレッド処理装置は、割り当てられたスレッドをそれぞれ実行する複数のプロセッサ部PE0~PEN-1と、各プロセッサ部PE0~PEN-1を制御してスレッドの実行を統括的に管理するスレッドマネージャ1と、各プロセッサ部PE0~PEN-1による処理結果等を格納するメモリ部2とを備える。スレッドマネージャ1は、各プロセッサ部にそれぞれ別個のスレッドを割り当てて、複数のプログラム経路を同時に投機的に実行させてプログラムの処理速度の向上を図る。また、スレッドマネージャ1は、スレッド間でデータ転送を行う必要が生じた場合には、各プロセッサ部内のレジスタ12あるいはロードストアバッファ14を用いてデータ転送を行う。



【特許請求の範囲】

【請求項 1】プログラムを複数のスレッドに分割し、これらスレッドを複数のプロセッサ部に割り当てて投機的に実行させる投機的マルチスレッド処理方法において、各プロセッサ部は、割り当てられたスレッドを投機的に実行する投機実行状態と、割り当てられたスレッドを確定的に実行する確定実行状態とを有し、各プロセッサ部の動作を制御するスレッドマネージャにより、投機状態にある少なくとも一部のプロセッサ部を用いて、プログラム内の複数の実行経路を同時に投機的に実行させることを特徴とする投機的マルチスレッド処理方法。

【請求項 2】前記スレッドマネージャは、各プロセッサ部によるスレッド実行開始、実行終了、および実行状態の切り替えと、異なるスレッド間でのデータ転送とを制御することを特徴とする請求項 1 に記載の投機的マルチスレッド処理方法。

【請求項 3】前記スレッドマネージャは、各プロセッサ部におけるスレッドの実行状況を示すプロセッサ状態表、各プロセッサ部のスレッド割り当て状況を示すスレッド実行表、およびスレッド間の起動関係を示すスレッド関係表を有し、これらの表を用いて、各プロセッサ部が実行するスレッドを決定するとともに、各プロセッサ部によるスレッド間のデータ転送を制御することを特徴とする請求項 1 または 2 に記載の投機的マルチスレッド処理方法。

【請求項 4】前記スレッドマネージャは、各プロセッサ部から通知されたスレッドの生成要求とスレッドの実行終了通知とを含む各種のスレッド制御要求コマンドを格納するFIFO型バッファを有し、このFIFO型バッファに格納された前記スレッド制御要求コマンドを順に読み出して前記プロセッサ部の動作を制御することを特徴とする請求項 1 ～ 3 のいずれかに記載の投機的マルチスレッド処理方法。

【請求項 5】前記スレッドマネージャは、投機実行状態でのスレッドの実行が誤りであることがわかった場合には、そのスレッドの実行と、そのスレッドから起動されるすべてのスレッドの実行とを中止することを特徴とする請求項 1 ～ 4 のいずれかに記載の投機的マルチスレッド処理方法。

【請求項 6】各プロセッサ部は、複数ビットからなるレジスタを有し、各レジスタは、異なるプロセッサ部間でレジスタ内のデータ転送を行う場合に、送信側プロセッサ部から転送されたレジスタ内のデータが受信側プロセッサ部のレジスタに格納されたか否かを示す識別ビットを有し、受信側プロセッサ部は、前記識別ビットの値により、レジスタ内のデータ転送が終了したか否かを判断することを特徴とする請求項 1 ～ 5 のいずれかに記載の投機的マルチスレッド処理方法。

【請求項 7】各プロセッサ部は、メモリに書き込むべきデータ、またはメモリから読み出されたデータを一時的に格納するロードストアバッファを有し、

前記ロードストアバッファは、送信側プロセッサ部内の前記ロードストアバッファから転送されたデータが受信側プロセッサ部内の前記ロードストアバッファに格納されたか否かを示す識別ビットを有し、

受信側プロセッサ部は、前記識別ビットの値により、前記ロードストアバッファのデータ転送が終了したか否かを判断することを特徴とする請求項 1 ～ 6 のいずれかに記載の投機的マルチスレッド処理方法。

【請求項 8】各プロセッサ部が有する前記ロードストアバッファは順次に接続され、

前記ロードストアバッファは、各プロセッサ部で実行中のスレッドを起動した親スレッドに対応するプロセッサ部のID番号と、各プロセッサ部で実行中のスレッドが起動した子スレッドに対応するプロセッサ部のID番号とを格納し、投機実行状態でロード命令が通知された場合には、親スレッドに対応するロードストアバッファを参照し、投機実行状態でストア命令が通知された場合には、子スレッドに対応するロードストアバッファに内容を転送することを特徴とする請求項 7 に記載の投機的マルチスレッド処理方法。

【請求項 9】プログラムを複数のスレッドに分割し、これらスレッドを複数の各プロセッサ部に割り当てて投機的に実行させる投機的マルチスレッド処理方法において、

各プロセッサ部は、割り当てられたスレッドを投機的に実行する投機実行状態と、割り当てられたスレッドを確定的に実行する確定実行状態とを有し、プロセッサ部間の通信により、他のプロセッサ部によるスレッド実行開始、実行終了、および実行状態の切り替えと、異なるスレッド間でのデータ転送とを制御し、投機実行状態にある少なくとも一部のプロセッサ部を用いて、プログラム内の複数の実行経路を同時に投機的に実行させることを特徴とする投機的マルチスレッド処理方法。

【請求項 10】プログラムを複数のスレッドに分割し、これらスレッドを複数の各プロセッサ部に割り当てて投機的に実行させる投機的マルチスレッド処理装置において、

各プロセッサ部には、割り当てられたスレッドを投機的に実行する投機実行状態と、割り当てられたスレッドを確定的に実行する確定実行状態とが設けられ、投機実行状態にある少なくとも一部のプロセッサ部がプログラム内の複数の実行経路を同時に投機的に実行するように各プロセッサ部を制御するスレッドマネージャを備えることを特徴とする投機的マルチスレッド処理装置。

【請求項 11】プログラムを複数のスレッドに分割し、

これらスレッドを複数のプロセッサ部に割り当てて投機的に実行させる投機的マルチスレッド処理装置において、

各プロセッサ部には、割り当てられたスレッドを投機的に実行する投機実行状態と、割り当てられたスレッドを確定的に実行する確定実行状態とが設けられ、各プロセッサ部は、他のプロセッサ部と通信を行った結果により、他のプロセッサ部のスレッド実行開始、実行終了、動作状態の切り替え、および異なるスレッド間でのデータ転送に関する情報を格納する記憶部と、投機実行状態のときにプログラム内の複数の実行経路が同時に投機的に実行されるように、対応するプロセッサ部を制御する制御部と、を有することを特徴とする投機的マルチスレッド処理装置。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、複数のプロセッサ部を有するマルチプロセッサシステムにおいて、プログラムの実行経路を事前に予測して投機的に実行する技術に関する。

【0002】

【従来の技術】複数のプロセッサ部を用いた並列処理の手法の中に、マルチスレッド手法と呼ばれるものがある。この手法は、一つのプログラムを複数のスレッドに分割し、各スレッドにそれぞれ異なるプロセッサ部を割り当てて並列実行させることにより、プログラムの実行を高速化するものである。ここでのスレッドは、個々の命令でもよいし、複数の命令でもよい。

【0003】一般に、スレッド間に依存関係がある場合には、依存関係が解消するまで並列実行を抑止する方策がとられる。例えば、スレッドBの実行がスレッドAの実行結果に依存する場合には、スレッドAの実行が終了するまでスレッドBの実行が抑止される。

【0004】これに対して、近年、スレッド間の依存関係の解消を待たずにスレッドを並列実行する投機的マルチスレッド処理方式が提案された。この方式では、プログラムの実行経路の中から1つを選択し、その選択経路（トレース）上の各スレッドについて、そのスレッドと依存関係のあるスレッドの実行終了を待たずに実行を開始する。

【0005】図24はあるプログラムにおけるスレッドの実行手順を示した流れ図である。また、図25は図24の流れ図を4台のプロセッサ部で並列実行した場合の従来方式によるスレッドの投機的な実行状況を示す図であり、図25の縦方向は時間軸を表している。なお、各プロセッサ部の実行状態は、スレッドマネージャにより制御される。

【0006】図24の流れ図において、例えば、（A、B、C、E）の経路を投機的に実行し、この経路が実際に実行される経路と一致したとする。この場合、まず、

図25に示すように、時刻 t_0 でプロセッサ部PE0にスレッドAを割り当てる。プロセッサ部PE0は、スレッドAの実行の途中（図25の時刻 t_1 ）に、スレッドBの実行要求をスレッドマネージャに通知する。

【0007】スレッドマネージャは、スレッドBの実行要求（fork）を受けると、プロセッサ部PE1にスレッドBを割り当てる。これにより、プロセッサ部PE1は、スレッドBの実行を投機状態で開始する。

【0008】プロセッサ部PE1は、スレッドBの実行の途中（図25の時刻 t_2 ）に、スレッドCの実行要求（fork）をスレッドマネージャに通知する。スレッドマネージャは、スレッドCの実行要求（fork）を受けると、プロセッサ部PE2にスレッドCを割り当てる。これにより、プロセッサ部PE2は、スレッドCの実行を投機状態で開始する。

【0009】プロセッサ部PE2は、スレッドCの実行の途中（図25の時刻 t_3 ）に、スレッドEの実行要求（fork）をスレッドマネージャに通知する。スレッドマネージャは、スレッドEの実行要求（fork）を受けると、プロセッサ部PE3にスレッドEを割り当てる。これにより、プロセッサ部PE3は、スレッドEの実行を投機状態で開始する。

【0010】また、プロセッサ部PE0は、スレッドAの実行の途中（図25の時刻 t_3 ）にm、スレッドBの実行が確定（commit）したことをスレッドマネージャに通知する。これを受けて、スレッドマネージャは、スレッドBを実行中のプロセッサ部PE1に対して、プロセッサ部PE1が確定したことを知らせる。これを受けて、プロセッサ部PE1は、スレッドBを確定状態に遷移させて実行する。

【0011】プロセッサ部PE1は、スレッドBの実行の途中（図25の時刻 t_4 ）に、スレッドCの実行が確定（commit）したことをスレッドマネージャに通知する。これを受けて、スレッドマネージャは、スレッドCを実行中のプロセッサ部PE2に対して、プロセッサ部PE2が確定したことを知らせる。これを受けて、プロセッサ部PE2は、スレッドCを確定状態に遷移させて実行する。

【0012】プロセッサ部PE2は、スレッドCの実行の途中（図25の時刻 t_5 ）に、スレッドDの実行が確定（commit）したことをスレッドマネージャに通知する。これを受けて、スレッドマネージャは、スレッドEを実行中のプロセッサ部PE3に対して、プロセッサ部PE3が確定したことを知らせる。これを受けて、プロセッサ部PE3は、スレッドEを確定状態に遷移させて実行する。

【0013】以上のように、各プロセッサ部はスレッドを次々に起動し、各スレッドは投機状態から確定状態に遷移しながら実行される。投機的に実行したプログラム実行経路が実際の実行経路と一致する確率が高ければ、プログラムを高速に処理することが可能となり、並列処理による効率化が図れる。

【0014】

【発明が解決しようとする課題】一方、図26は、図24と同様のプログラムに対して投機的な処理を行った結果、投機的に実行したプログラム経路と実際の実行経路とが一致しなかった例を示す図である。図26の時刻 $t_0 \sim t_3$ では、図25と同様の処理が行われる。プロセッサ部PE1は、スレッドBの実行の途中（図26の時刻 t_4 ）に、スレッドCの実行が誤っていたため、処理を中止（kill）することをスレッドマネージャに通知する。これを受けて、スレッドマネージャは、スレッドCを実行しているプロセッサ部PE2とスレッドCにより起動されたスレッドEを実行しているPE3とに対して、実行の中止を通知する。これを受けて、プロセッサ部PE2、PE3は、それぞれスレッドC、Eの実行を中止する。

【0015】その後、プロセッサ部PE1は、スレッドBの実行の途中（図26の時刻 t_6 ）に、スレッドDの実行要求（fork）をスレッドマネージャに通知する。スレッドマネージャは、スレッドDの実行要求（fork）を受けると、プロセッサ部PE2にスレッドDを割り当てる。これにより、プロセッサ部PE2は、スレッドDの実行を投機状態で開始する。

【0016】その後、プロセッサ部PE2は、スレッドDの実行の途中（図26の時刻 t_7 ）に、スレッドHの実行要求（fork）をスレッドマネージャに対して行う。スレッドマネージャは、スレッドHの実行要求（fork）を受けると、プロセッサ部PE3にスレッドHを割り当てる。これにより、プロセッサ部PE3は、スレッドHの実行を投機状態で開始する。

【0017】その後、時刻 t_8 、 t_9 において、プロセッサ部PE1、PE2はそれぞれ、スレッドD、Hの実行が確定（commit）したことをスレッドマネージャに通知する。このように、投機的な実行処理を行う場合、プログラムの予測経路が的中する確率が高ければ効率よくプログラムを実行できるが、予測経路が外れる確率が高い場合には、予測処理した経路の実行を中止して、いったん後戻りして別経路を取らなければならず、プログラム全体の実行効率を向上できなくなる。

【0018】また、従来の投機的実行方式では、複数のプログラム経路を同時に実行するような処理を行っていなかったため、単一のプログラム経路の実行確率が高くない場合には、予測が外れる可能性が高く、プログラムを高速に処理できないという問題がある。

【0019】本発明は、このような点に鑑みてなされたものであり、その目的は、プログラムの処理速度を向上できる投機的マルチスレッド処理方法および投機的マルチスレッド処理装置を提供することにある。

【0020】

【課題を解決するための手段】上述した課題を解決するために、本発明は、プログラムを複数のスレッドに分割

し、これらスレッドを複数のプロセッサ部に割り当てて投機的に実行させる投機的マルチスレッド処理方法において、各プロセッサ部は、割り当てられたスレッドを投機的に実行する投機実行状態と、割り当てられたスレッドを確定的に実行する確定実行状態とを有し、各プロセッサ部の動作を制御するスレッドマネージャにより、投機状態にある少なくとも一部のプロセッサ部を用いて、プログラム内の複数の実行経路を同時に投機的に実行させる。

【0021】また、本発明は、プログラムを複数のスレッドに分割し、これらスレッドを複数の各プロセッサ部に割り当てて投機的に実行させる投機的マルチスレッド処理方法において、各プロセッサ部は、割り当てられたスレッドを投機的に実行する投機実行状態と、割り当てられたスレッドを確定的に実行する確定実行状態とを有し、プロセッサ部間の通信により、他のプロセッサ部によるスレッド実行開始、実行終了、および実行状態の切り替えと、異なるスレッド間でのデータ転送とを制御し、投機実行状態にある少なくとも一部のプロセッサ部を用いて、プログラム内の複数の実行経路を同時に投機的に実行させる。

【0022】また、本発明は、プログラムを複数のスレッドに分割し、これらスレッドを複数の各プロセッサ部に割り当てて投機的に実行させる投機的マルチスレッド処理装置において、各プロセッサ部には、割り当てられたスレッドを投機的に実行する投機実行状態と、割り当てられたスレッドを確定的に実行する確定実行状態とが設けられ、投機実行状態にある少なくとも一部のプロセッサ部がプログラム内の複数の実行経路を同時に投機的に実行するように各プロセッサ部を制御するスレッドマネージャを備える。

【0023】また、本発明は、プログラムを複数のスレッドに分割し、これらスレッドを複数のプロセッサ部に割り当てて投機的に実行させる投機的マルチスレッド処理装置において、各プロセッサ部には、割り当てられたスレッドを投機的に実行する投機実行状態と、割り当てられたスレッドを確定的に実行する確定実行状態とが設けられ、各プロセッサ部は、他のプロセッサ部と通信を行った結果により、他のプロセッサ部のスレッド実行開始、実行終了、動作状態の切り替え、および異なるスレッド間でのデータ転送に関する情報を格納する記憶部と、投機実行状態のときにプログラム内の複数の実行経路が同時に投機的に実行されるように、対応するプロセッサ部を制御する制御部と、を有する。

【0024】

【発明の実施の形態】以下、本発明に係る投機的マルチスレッド処理方法およびその装置について、図面を参照しながら具体的に説明する。

【0025】（第1の実施形態）図1は本発明に係る投機的マルチスレッド処理装置の第1の実施形態の概略構

成を示すブロック図である。図1の投機的マルチスレッド処理装置は、割り当てられたスレッドをそれぞれ実行する複数のプロセッサ部PE0～PE_{n-1}と、各プロセッサ部PE0～PE_{n-1}を制御してスレッドの実行を管理するスレッドマネージャ1と、各プロセッサ部PE0～PE_{n-1}による処理結果等を格納するメモリ部2とを備える。

【0026】各プロセッサ部PE0～PE_{n-1}は、スレッドマネージャ1から割り当てられたスレッドを実行し、スレッドの実行開始/終了、スレッドの実行に伴って生じる他のスレッドの実行、および他のスレッドとの通信や同期などの各種のスレッド制御要求をスレッドマネージャ1に通知する。スレッドマネージャ1は、各プロセッサ部PE0～PE_{n-1}から要求された各種のコマンドに基づいて各プロセッサ部PE0～PE_{n-1}を制御する。メモリ部2は、各プロセッサ部PE0～PE_{n-1}からアクセス可能なように接続されている。

【0027】図2は各プロセッサ部PE0～PE_{n-1}の内部構成を示すブロック図である。図2に示すように、各プロセッサ部PE0～PE_{n-1}は、プロセッサコア11と、レジスタ12と、キャッシュメモリ13と、ロードストアバッファ14とを有する。プロセッサコア11は、例えば通常のRISCプロセッサコアからなり、スーパースカラ・プロセッサでも、VLIWプロセッサでもよい。プロセッサコア11とレジスタ12はスレッドマネージャ1に接続されており、各プロセッサ部PE0～PE_{n-1}内のロードストアバッファ14は相互に接続されている。

【0028】プロセッサ部内のレジスタ12は、図3に詳細構成を示すように、レジスタA12aとレジスタB12bの2つからなる。レジスタA12aは通常のスレッド実行用として用いられ、レジスタB12bは待避用として用いられる。レジスタB12bを設ける理由は、スレッドの切り替えを高速化するためである。これらレジスタ12a、12bには、同期ビットが1ビットずつ設けられる。この同期ビットは、スレッドマネージャ1との同期をとるためのものであり、同期ビットの値により、スレッドマネージャ1はレジスタ内のデータ転送が行われたか否かを判断する。

【0029】図4はスレッドマネージャ1の内部構成を示すブロック図である。図4に示すように、スレッドマネージャ1は、プロセッサ状態表21と、スレッド実行表22と、スレッド関係表23と、コンテキストバッファ24と、制御部25と、FIFO型バッファ26と、レジスタ間転送部27とを有する。このうち、プロセッサ状態表21、スレッド実行表22およびスレッド関係表23は、例えばROMテーブルで構成される。

【0030】プロセッサ状態表21は、各プロセッサ部PE0～PE_{n-1}の実行状態と、実行中のスレッドのID番号と、実行コンテキストのID番号との関係を表にしたものであり、この表により、各プロセッサ部PE0～PE_{n-1}の実行状態を把握できる。各プロセッサ部PE0～PE_{n-1}

-1の実行状態には、例えば、スレッドを確定的に実行する確定実行状態、スレッドを投機的に実行する投機実行状態、およびスレッドの実行を見合わせる待機状態の3種類がある。

【0031】スレッド実行表22は、スレッドの実行状態と、各スレッドを実行しているプロセッサ部の番号との関係を表にしたものであり、この表により、各スレッドを実行しているプロセッサ部を把握できる。

【0032】スレッド関係表23は、スレッドの起動に関する情報を表にしたものであり、この表により、親スレッドと子スレッドとの関係を把握できる。

【0033】コンテキストバッファ24には、各プロセッサ部PE0～PE_{n-1}内のレジスタ12の値が格納される。コンテキストバッファ24は、レジスタ12の読み出しや書き込みを直接行うことができ、すべてのレジスタ12のデータをコピーする場合等に用いられる。

【0034】FIFO型バッファ26は、各プロセッサ部PE0～PE_{n-1}がスレッドマネージャ1に対して通知する各種のスレッド制御要求コマンドを格納する。スレッドマネージャ1は、FIFO型バッファ26に格納されたコマンドを順に読み出して、コマンドに応じた制御を行う。

【0035】図5は各プロセッサ部PE0～PE_{n-1}の内部に設けられるロードストアバッファ14(LSB)の詳細構成を示すブロック図である。ロードストアバッファ14は、投機実行状態のときに、スレッド実行によるメモリの整合性を保持することを目的とする。ロードストアバッファ14には、図5に示すように、親スレッドを実行するプロセッサ部のID番号31と、子スレッドを実行するプロセッサ部のID番号32と、確定実行状態か投機実行状態かの実行状態を判定する判定ビット33とが格納される。この他、ロードストアバッファ14には、バッファエントリ情報として、ロードかストアかを識別するロードストア識別子34と、親スレッドからの書き込み確認の同期ビット35と、ロードストアの対象アドレス36と、ストア時のデータ37とが格納される。

【0036】図6は図1の投機的マルチスレッド処理装置のプログラム実行例を示す図である。図6は、図24のプログラムを実行する際、複数のプログラム経路を同時に実行する例を示している。

【0037】図6の時刻t0、t1では、図25の時刻t0、t1と同様の処理を行う。プロセッサ部PE0は、スレッドBの実行の途中(図6の時刻t2)に、スレッドBの実行が確定(commit)したことをスレッドマネージャ1に通知する。これを受けて、スレッドマネージャ1は、スレッドBを実行中のプロセッサ部PE1に対して、スレッドBが確定したことを知らせる。これを受けて、プロセッサ部PE1は、スレッドBを確定状態に遷移させて実行する。

【0038】また、プロセッサ部PE1は、スレッドBの

実行の途中（図6の時刻 t_2 ）に、スレッドC、Dの実行要求（fork）をスレッドマネージャ1に対して行う。これを受けて、スレッドマネージャ1は、プロセッサ部PE2にスレッドCを、プロセッサ部PE3にスレッドDを割り当てる。これにより、プロセッサ部PE1、PE2はそれぞれ、スレッドC、Dの実行を投機状態で開始させる。

【0039】その後、プロセッサ部PE1は、スレッドC、Dの実行の途中（図6の時刻 t_3 ）に、スレッドCの実行が確定（commit）したことと、スレッドDの実行が誤りであったことをスレッドマネージャ1に通知する。これを受けて、スレッドマネージャ1は、プロセッサ部PE2にスレッドCの確定を通知するとともに、プロセッサ部PE3にスレッドDの中止を通知する。これを受けて、プロセッサ部PE2はスレッドCを確定実行状態に遷移させて実行し、プロセッサ部PE3はスレッドDの実行を中止する。

【0040】その後、プロセッサ部PE2は、スレッドCの実行の途中（図6の時刻 t_4 ）に、スレッドE、Fの実行要求（fork）をスレッドマネージャ1に対して行う。これを受けて、スレッドマネージャ1は、プロセッサ部PE0にスレッドEを、プロセッサ部PE1にスレッドFを割り当てる。これにより、プロセッサ部PE0、PE1はそれぞれ、スレッドE、Fの実行を開始させる。

【0041】その後、プロセッサ部PE2は、スレッドFの実行の途中（図6の時刻 t_5 ）に、スレッドFの実行が誤りであったことをスレッドマネージャ1に通知する。これを受けて、スレッドマネージャ1は、プロセッサ部PE1にスレッドFの中止を通知する。これを受けて、プロセッサ部PE1はスレッドFの実行を中止する。

【0042】このように、本実施形態では、複数のプロセッサ部が異なるプログラム経路を同時に投機的に実行するため、1つのプログラム経路のみについて投機的な実行を行う場合に比べて、プログラムの実行効率を向上できる。

【0043】図7は図6の時刻 t_2 、 t_3 、 t_4 におけるスレッドマネージャ1の内部状態を示す図である。まず、時刻 t_2 において、制御部25は、スレッドA、Bを確定状態で実行し、スレッドC、Dを投機状態で実行することをプロセッサ状態表21に設定する。また、制御部25は、スレッドA～Dをそれぞれプロセッサ部PE0～PE3で実行することをスレッド実行表22に設定する。さらに、制御部25は、スレッドAからスレッドBを起動し、スレッドBからスレッドCを起動し、スレッドBからスレッドDを起動することをスレッド実行表22に設定する。時刻 t_3 において、制御部25は、スレッドBを確定状態で実行し、スレッドCを投機状態で実行することをプロセッサ状態表21に設定する。また、制御部25は、スレッドBをプロセッサ部PE1で実行し、スレッドCをプロセッサ部PE2で実行することをス

レッド実行表22に設定する。さらに、制御部25は、スレッドBからスレッドCを起動することをスレッド関係表23に設定する。

【0044】時刻 t_4 において、制御部25は、スレッドE、Fを投機状態で実行し、スレッドCを確定状態で実行することをプロセッサ状態表21に設定する。また、制御部25は、スレッドEをプロセッサ部PE0で、スレッドFをプロセッサ部PE1で、スレッドCをプロセッサ部PE2で実行することをスレッド実行表22に設定する。また、制御部25は、スレッドCからスレッドE、Fを起動することをスレッド実行表22に設定する。

【0045】図8は図6の時刻 t_2 におけるスレッドマネージャ1の動作を模式的に示した図である。制御部25は、図示のようなプロセッサ状態表21とスレッド実行表22を生成し、これらの表に基づいて、プロセッサ部PE2にスレッドCを実行させ、プロセッサ部PE3にスレッドDを実行させる。

【0046】また、FIFO型バッファ26には、プロセッサ部PE1からのスレッド制御要求コマンドが格納される。具体的には、スレッドCの実行を確定し、スレッドDの実行を中止する旨のスレッド制御要求コマンドがFIFO型バッファ26に格納される。

【0047】図9は各プロセッサ部PE0～PE $n-1$ がスレッドマネージャ1に対して通知するコマンドの内容を説明する図である。図9の(1)に示すforkコマンドは、スレッドT1からスレッドT2の起動を要求するコマンドである。(2)に示すtermコマンドは、自己のスレッドの実行終了を通知するコマンドである。(3)に示すkillコマンドは、スレッドT1からスレッドT2を強制終了させることを要求するコマンドである。(4)に示すcommitコマンドは、スレッドT1からスレッドT2に対して、スレッドT2の投機実行状態から確定実行状態への遷移を通知するコマンドである。(5)に示すsendコマンドは、スレッドT1からスレッドT2に対してレジスタ12の内容を転送するコマンドである。(6)に示すreceiveコマンドは、スレッドT1からの転送データをスレッドT2で受信するコマンドである。(7)に示すcheckコマンドは、スレッドT1からスレッドT2へのメモリ転送が完了したか否かをチェックするコマンドである。(8)に示すnotifyコマンドは、スレッドT1からスレッドT2にメモリのアドレスとデータを転送するコマンドである。

【0048】各プロセッサ部PE0～PE $n-1$ が図9のいずれかのコマンドをスレッドマネージャ1に対して通知すると、スレッドマネージャ1内の制御部25は、図10に示す処理を行う。以下、図10のフローチャートに基づいて、制御部25の動作を説明する。

【0049】まず、制御部25は、ステップS1に示すように、各プロセッサ部PE0～PE $n-1$ から通知されたコ

マンドを格納しているFIFO型バッファ26の内容を順に読み出す。読み出されたコマンドがforkコマンドの場合にはステップS2のfork処理を、termコマンドの場合にはステップS3のterm処理を、killコマンドの場合にはステップS4のkill処理を、commitコマンドの場合にはステップS5のcommit処理を、sendコマンドの場合にはステップS6のsend処理を、receiveコマンドの場合にはステップS7のreceive処理を、notifyコマンドの場合にはステップS8のnotify処理を、checkコマンドの場合にはステップS9のcheck処理を行う。

【0050】図11は図10のステップS2のfork処理を具体的に説明するフローチャートである。まず、プロセッサ状態表21に基づいて空きプロセッサ部を検索し（ステップS11）、空きプロセッサ部があれば、そのプロセッサ部のID番号を変数Kに代入する。

【0051】次に、空きプロセッサ部（PE）が存在するかどうかを判定し（ステップS12）、空きプロセッサ部が存在すれば、プロセッサ状態表21のK行目に、forkコマンドの引数T2に対応するプロセッサ部を登録する（ステップS13）。

【0052】次に、スレッド実行表22のK行目に、引数T2に対応するプロセッサ部を登録する（ステップS14）。次に、引数T1、T2に対応する両プロセッサ部の起動関係をスレッド関係表23に登録する（ステップS15）。次に、スレッドT1を実行するプロセッサ部のレジスタ12の内容を、スレッドT2を実行するプロセッサ部のレジスタ12にコピーする（ステップS16）。次に、スレッド実行表22のK行目に設定されたプロセッサ部にスレッドT2の実行開始を指示する。

【0053】一方、ステップS12で空きプロセッサ部がないと判定された場合には、空きプロセッサ部が見つかるまで、プロセッサ待ちバッファに登録される（ステップS18）。次に、スレッドT1を実行するプロセッサ部のレジスタ12の内容をコンテキストバッファ24にコピーする（ステップS19）。次に、スレッドT1からスレッドT2を起動することをスレッド関係表23に設定する（ステップS20）。

【0054】図12は図10のステップS3のterm処理を具体的に説明するフローチャートである。まず、スレッド実行表22に基づいて、スレッドT1に対応するプロセッサ部PEを検索し、検索されたプロセッサ部に対応するID番号を変数Kに代入する（ステップS21）。

【0055】次に、プロセッサ状態表21の中から、変数Kに対応する行を削除する（ステップS22）。次に、スレッド実行表22の中から、変数Kに対応する行を削除する（ステップS23）。次に、スレッド関係表23の中から、スレッドT1に基づいて起動される全スレッドを削除する（ステップS24）。

【0056】図13は図10のステップS4のkill処理を具体的に説明するフローチャートである。まず、スレ

ッド実行表22に基づいて、スレッドT2を実行中のプロセッサ部を検索する（ステップS41）。次に、プロセッサ部PE（K）に実行中止の指示を送信する（ステップS42）。次に、プロセッサ状態表21のK行目を削除する（ステップS43）。次に、スレッド実行表22のK行目を削除する（ステップS44）。次に、スレッド関係表23のスレッドT1、T2を削除する（ステップS45）。次に、スレッド関係表23からスレッドT2の子孫を検索する（ステップS46）。次に、スレッドT2の子孫に対して同様の処理を行う（ステップS47）。

【0057】図14は図10のステップS5のcommit処理を具体的に説明するフローチャートである。まず、スレッド実行表22に基づいて、スレッドT2を実行しているプロセッサ部を検索し、検索されたプロセッサ部に対応するID番号を変数Kに代入する（ステップS61）。

次に、変数Kに対応するプロセッサ部に対して、このプロセッサ部が実行しているスレッドの実行が確定した旨を通知する（ステップS62）。次に、プロセッサ状態表21を更新する（ステップS63）。すなわち、変数Kに対応するプロセッサ部が実行しているスレッドが確定実行状態になったことをプロセッサ状態表21に設定する。

【0058】図15は図10のステップS6のsend処理を具体的に説明するフローチャートである。まず、スレッドT1を実行しているプロセッサ部から、レジスタ12の番号とレジスタ12の値を受信する（ステップS81）。次に、スレッド実行表22に基づいて、スレッドT2を実行中のプロセッサ部を検索し、検索されたプロセッサ部に対応するID番号を変数Kに代入する（ステップS82）。次に、変数Kに対応するプロセッサ部のレジスタ12にデータを書き込むとともに、そのレジスタ12の同期ビットを「1」にする。

【0059】なお、プロセッサ部がスレッドマネージャ1に対してreceiveコマンドを通知した場合には、制御部25は特に何も処理は行わない。

【0060】図16は図10のステップS8のnotify処理を具体的に説明するフローチャートである。まず、スレッドT1を実行しているプロセッサ部から、メモリアドレスとメモリデータを受信する（ステップS101）。次に、スレッド実行表22に基づいて、スレッドT2を実行中のプロセッサ部を検索する（ステップS102）。次に、検索されたプロセッサ部PE（K）に対応するプロセッサ部のロードストアバッファ14に、ステップS101で受信したメモリアドレスとメモリデータを格納する（ステップS103）。

【0061】なお、プロセッサ部がスレッドマネージャ1に対してcheckコマンドを通知した場合には、制御部25は特に何も処理は行わない。

【0062】次に、異なるスレッド間でのレジスタ12の転送について説明する。スレッド間でデータ転送を行

うには、スレッドを同期させる必要がある。このため、本実施形態では、プロセッサ部内のレジスタ12に同期ビットを設けてスレッドを同期させる。

【0063】図17はスレッド間でのレジスタ12の転送を説明する図である。例えば、プロセッサ部PE1で実行中のスレッドBのレジスタ12から、プロセッサ部PE3で実行中のスレッドのレジスタ12にデータを転送する場合について説明する。プロセッサ部PE1がスレッドBを実行中にsendコマンドをスレッドマネージャ1に対して通知すると、スレッドマネージャ1は、通知されたsendコマンドの引数により、スレッドB、D間でレジスタ転送を行うことを認識する。また、スレッド実行表22により、スレッドDはプロセッサ部PE3が実行中であることを認識する。

【0064】スレッドマネージャ1のレジスタ間転送部27は、プロセッサ部PE1のレジスタ12の内容を受信し、その受信内容をプロセッサ部PE3のレジスタ12に転送する。

【0065】上述したように、各プロセッサ部PE0～PE $n-1$ のレジスタ12には、同期ビットが1ビット設けられている。同期ビットの初期値は「0」であり、他のスレッドから転送されたデータがレジスタ12に設定されたときに、対応するレジスタ12の同期ビットは「1」に設定される。

【0066】図17の場合、スレッドマネージャ1がプロセッサ部PE1のレジスタ12から転送されたデータをプロセッサ部PE3のレジスタ12に格納したときに、プロセッサ部PE3のレジスタ12の同期ビットは「1」に設定される。

【0067】各プロセッサ部PE0～PE $n-1$ は、スレッドD中のreceiveコマンドを実行するときに、同期ビットを参照する。例えば、同期ビットがすでに「1」であれば、すでにデータが転送されたと判断し、同期ビットをクリアして実行を続ける。一方、同期ビットが「0」であれば、まだスレッドBからデータが転送されていないと判断し、同期ビットが設定されるまでプログラムの実行を中断する。

【0068】次に、異なるスレッド間でのメモリデータの転送について説明する。ここで、メモリデータとは、メモリ部2に書き込むべきデータまたはメモリ部2から読み出したデータを指す。

【0069】図18はスレッド間でのメモリデータの転送を説明する図であり、プロセッサ部PE1のスレッドBからプロセッサ部PE3のスレッドDにメモリデータを転送する例を示している。各プロセッサ部PE0～PE $n-1$ にはそれぞれロードストアバッファ14が設けられており、このロードストアバッファ14には、各プロセッサ部PE0～PE $n-1$ がメモリ部2に書き込むべきデータとそのアドレス情報、または各プロセッサ部PE0～PE $n-1$ がメモリ部2から読み出したデータとそのアドレス情報が

格納される。

【0070】具体的には、スレッドが投機実行状態のときには、メモリ部2への書き込みは行われず、ロードストアバッファ14内に結果が保持される。スレッドが投機実行状態から確定実行状態に移したとき、ロードストアバッファ14の内容がメモリに書き込まれ、スレッドの実行が無効化されたときはロードストアバッファ14が初期化される。

【0071】一方、メモリの読み出しを行う場合は、スレッドが確定実行状態であっても、投機実行状態であっても、実際にメモリにアクセスしてデータの読み出しを行う。

【0072】例えば、図18は、プロセッサ部PE1がメモリの100番地にデータ「1234」を書き込み、この番地のデータをプロセッサ部PE3が読み出す例を示している。プロセッサ部PE1は、投機実行状態のときは、メモリにデータを書き込む代わりに、ロードストアバッファ14にデータ「1234」を書き込む。プロセッサ部PE1、PE3はそれぞれ別個にロードストアバッファ14を有するため、プロセッサ部PE1のロードストアバッファ14に格納されたデータは、制御部25を介してプロセッサ部PE3のロードストアバッファ14に転送される。

【0073】ところで、ソースプログラムをコンパイルするときにメモリアクセス時のメモリアドレスがわかっている場合と、プログラムを実行する時点で初めてメモリアドレスがわかる場合とがある。後者は、例えば、間接アドレス等を利用している場合等である。前者と後者でメモリデータの転送手法が異なるため、以下、場合分けして説明する。

【0074】メモリアクセス時のメモリアドレスが予めわかっている場合には、スレッドBは、スレッドDがメモリの100番地をアクセスすることをコンパイル時に認識する。そこで、100番地にストアすべきデータをロードストアバッファ14に格納した直後に、格納したことを知らせるnotifyコマンドをスレッドマネージャ1に通知する。その後、100番地の内容をスレッドマネージャ1を経由して、スレッドDを実行中のプロセッサ部PE3のロードストアバッファ14に転送する。

【0075】一方、スレッドDは、100番地へのアクセスはスレッドBがロードストアバッファ14に格納したデータであることをコンパイル時に認識する。そこで、100番地のデータをロードする命令をスレッドマネージャ1に通知する直前にcheckコマンドを通知する。

【0076】スレッドDがcheckコマンドを実行する前に、スレッドBからのデータがロードストアバッファ14に到着していれば、checkコマンドを飛ばして次のロード命令を実行する。checkコマンドの実行前にスレッドBからのデータがロードストアバッファ14に到着していないときには、データの到着を待つ。データが到着したか否かを検出するため、ロードストアバッファ14

に格納される各データには1ビットの同期ビットが設けられる。

【0077】スレッドマネージャ1は、プロセッサ部PE1がnotifyコマンドを通知したときに、ロードストアバッファ14の対応する番地のデータ中の同期ビットを「1」に設定する。プロセッサ部は、checkコマンドの実行時に同期ビットの値を判定して、転送データが到達したか否かを認識する。

【0078】一方、メモリアクセスの際のメモリアドレスがプログラム実行時に初めてわかる場合には、ロードストアバッファ14は、プロセッサ部にスレッドを割り当てて、スレッドの実行開始時にスレッドマネージャ1から親スレッドのプロセッサ部のIDを取得する。また、子スレッドを実行するプロセッサ部のIDは、スレッドがforkなどのスレッド制御命令を実行するたびにスレッドマネージャ1を経由してロードストアバッファ14に転送される。

【0079】ロードストアバッファ14は、各プロセッサ部PE0～PE_{n-1}の実行状態とメモリアクセスの種類により、以下のような動作を行う。

【0080】(a)確定実行状態のプロセッサ部からメモリのロード要求があった場合
この場合、何もせずにロードストアバッファ14よりも下のメモリ階層、すなわちメモリ部2から実際にデータをロードする。

【0081】(b)確定実行状態のプロセッサ部からメモリのストア要求があった場合
メモリ部2にデータを書き込むと同時に、投機実行状態にある子スレッドのプロセッサ部のロードストアバッファ14にメモリアドレスと書き込みデータを転送する。

【0082】(c)投機実行状態のプロセッサ部からメモリのロード要求があった場合
ロードストアバッファ14内に、該当するアドレスと一致するデータがあれば、そのデータをプロセッサ部に渡す。一致するデータがなければ、まず、親スレッドのプロセッサ部のロードストアバッファ14を検索し、検索されたデータをプロセッサ部に渡すとともに、検索されたデータとアドレスを、自己のロードストアバッファ14内に1エントリを割り当てて格納する。仮に、エントリの空きがなければ、空きができるまでロード処理の実行を停止する。なお、ロードストアバッファ14のエントリは、確定実行状態への移行により解放される。投機実行状態から確定実行状態に移行すると、ロードストアバッファ14内のすべてのデータをメモリ部2に格納した後、エントリを解放する。すなわち、ロードストアバッファ14のエントリを越える数の投機的なメモリアクセスは行えない。また、エントリを解放すると、同期ビットはリセットされる。

【0083】(d)投機実行状態のプロセッサ部からメモリのロード要求があった場合

ロードストアバッファ14内に該当するデータが存在するか否かに関係なく、ロードストアバッファ14に1エントリを割り当ててデータを格納し、かつ、子スレッドのプロセッサ部のロードストアバッファ14にもデータを転送して格納する。もしもエントリの空きがなければ、空きができるまでロード処理の実行を停止する。この場合、同期ビットはセットしない。また、ロードストアバッファ14は、他のプロセッサ部のロードストアバッファ14からの要求により、次の動作を行う。

【0084】(a)他のロードストアバッファ14からデータのロードの問い合わせが来た場合

これは、子スレッドから親スレッドにデータを要求する場合である。自己のロードストアバッファ14が該当データを所有している場合、すなわち、過去に投機実行状態でロードストアバッファ14にデータを格納した場合には、そのデータを要求元に転送する。一方、自己のロードストアバッファ14が該当データを所有していない場合、自己が確定実行状態か投機実行状態かにより、処理が異なる。確定実行状態であれば、2次キャッシュから読み込んだデータを要求元に転送する。また、投機実行状態であれば、自己の親スレッドのプロセッサ部のロードストアバッファ14に問い合わせを行い、返ってきたデータを要求元に転送する。

【0085】(b)他のロードストアバッファ14からストアデータが転送されてきた場合

これは、親スレッドから子スレッドにデータを転送する場合である。自己のロードストアバッファ14が、転送されて来たデータと同一アドレスへのアクセスをすでに行ったか否かを、ロードストアバッファ14内の各エントリと比較して判別する。同一アドレスへのアクセスをすでに行っている場合には、プロセッサ部に対して実行のやり直しを通知する。すなわち、同時に起こったエントリにデータを書き込み、同時に同期ビットをセットする。エントリ内に同一アドレスのエントリが存在しない場合には、1エントリを割り当ててデータを格納し、同時に同期ビットをセットする。また、送られてきたデータは、エントリ内に該当アドレスがあるか否かにかかわらず、子スレッドに転送される。

【0086】図19は投機実行状態のロードが発生した場合の例であり、子スレッドがロード命令を実行しようとした状態を示している。図19では、3つのスレッドA、B、Cが動作中であり、スレッドAがスレッドBの親、スレッドBがスレッドCの親である。また、スレッドAは確定状態で実行され、スレッドB、Cは投機状態で実行されているものとする。

【0087】スレッドCを実行中のプロセッサ部においてロード命令が発生した場合、スレッドCのロードストアバッファ14には該当するエントリがないため、スレッドCの親であるスレッドBに問い合わせる。スレッドAは確定実行状態なので、スレッドAのロードストアバ

ッファ14にエントリが存在するならばその値を、エントリが存在しなければメモリ部2をアクセスして値をスレッドBに返す。スレッドBは、スレッドAから受け取った値をロードストアバッファ14に格納するとともに、スレッドCに返す。スレッドCは、スレッドBから受け取った値をロードストアバッファ14に格納するとともにプロセッサ部に返す。

【0088】一方、図20は投機実行状態のストアが発生した場合の例であり、親スレッドがストア命令を実行しようとした状態を示している。スレッドA、B、Cの親子関係や実行状態は図と同じである。

【0089】スレッドAを実行中のプロセッサ部にてストア命令が発生した場合、スレッドA内のロードストアバッファ14に1エントリを割り当ててアドレスとデータを格納するとともに、子スレッドBを実行中のプロセッサ部にアドレスとデータを転送する。スレッドBを実行するプロセッサ部は、自己のロードストアバッファ14内に1エントリを割り当てて転送されてきたデータを格納するとともに、子スレッドCにアドレスとデータを転送する。子スレッドCを実行するプロセッサ部は、自己のロードストアバッファ14に1エントリを割り当てて転送されてきたデータを格納する。

【0090】(第2の実施形態)第1の実施形態は、スレッドマネージャ1により複数のプロセッサ部の動作を統括的に管理する例を説明したが、スレッドマネージャ1は必ずしも必要ではない。以下に説明する第2の実施形態は、スレッドマネージャ1を省略したものである。

【0091】図21は本発明に係る投機的マルチスレッド処理装置の第2の実施形態の概略構成を示すブロック図である。図21の投機的マルチスレッド処理装置は、割り当てられたスレッドをそれぞれ実行する複数のプロセッサ部PE0~PE $n-1$ と、各プロセッサ部PE0~PE $n-1$ の演算結果等を格納するメモリ部2とを備えており、スレッドマネージャは備えていない。各プロセッサ部PE0~PE $n-1$ は縦続接続されている。

【0092】図1の装置ではスレッドマネージャがスレッドの管理を統括的に行ったが、図21の装置では各プロセッサ部PE0~PE $n-1$ がスレッドの管理を分散して行う。ここでは、プロセッサ部間の結合として、2本の単方向リングを想定する。スレッドの制御はリングの方向に沿って行われる。各プロセッサ部PE0~PE $n-1$ は、単方向リングの2つの接続先それぞれに対して1つずつ不図示のフォークバッファを有する。フォークバッファ内には、スレッド制御に必要な情報が保持される。

【0093】以下では、6台のプロセッサ部PE0~PE5により構成されるマルチプロセッサシステムを用いて、図24の流れ図に沿って各スレッドを実行する例を説明する。図22は6台のプロセッサ部の接続関係を示す図である。図22に示すように、プロセッサ部PE0はプロセッサ部PE1、PE2と通信を行い、プロセッサ部PE1は

プロセッサ部PE2、PE3と通信を行い、プロセッサ部PE3はプロセッサ部PE4、PE5と通信を行い、プロセッサ部PE4はプロセッサ部PE5、PE0と通信を行い、プロセッサ部PE5はプロセッサ部PE0、PE1と通信を行う。

【0094】図23は各プロセッサ部PE0~PE5の実行状態が変化する様子を示す図である。まず、図23(a)に示すように、プロセッサ部PE0によりスレッドAを実行する。その後、図23(b)に示すように、スレッドAはスレッドBをフォーク(fork)する。スレッドBは、プロセッサ部PE0に隣接するプロセッサ部PE1で実行が開始される。その後、図23(c)に示すように、スレッドBはスレッドC、Dをフォークする。スレッドCはそれぞれ、プロセッサ部PE1と接続されているプロセッサ部PE2、PE3で実行が開始される。

【0095】その後、図23(d)に示すように、スレッドCはスレッドE、Fをフォークしようとするが、スレッドCを実行しているプロセッサ部PE2に隣接するプロセッサ部PE3はすでにスレッドDを実行中であるため、スレッドCはスレッドEのみをフォークする。このスレッドEは、プロセッサ部PE4で実行が開始される。フォークできなかったスレッドFは、プロセッサ部PE2内のフォークバッファに格納され、プロセッサ部PE3、PE4が空くのを待つ。

【0096】その後、図23(e)に示すように、スレッドDはスレッドG、Hをフォークしようとするが、プロセッサ部PE3に隣接するプロセッサ部PE4はスレッドEを実行中であるため、スレッドGのみをプロセッサ部PE5にフォークして実行を開始する。フォークできなかったスレッドHに関しては、プロセッサ部PE3内のバッファに格納され、隣接するプロセッサ部PE4、PE5が空くのを待つ。

【0097】その後、図23(f)に示すように、プロセッサ部PE0のスレッドAの実行が終了し、プロセッサ部PE0は空き状態になる。その後、図23(g)に示すように、プロセッサ部PE1が実行中のスレッドBは、スレッドCに対してcommitコマンドを通知するとともに、スレッドDに対してkillコマンドを通知した後、実行を終了する。一方、killコマンドを受信したプロセッサ部PE3が実行中のスレッドDは、自己がフォークしたスレッドGに対してkillコマンドを通知した後、実行を終了する。同様に、プロセッサ部PE5で実行中のスレッドGは、killコマンドを受信して実行を終了する。この結果、プロセッサ部PE1、PE3、PE5が新たに空き状態になる。

【0098】その後、図23(g)に示すように、プロセッサ部PE2のスレッドC内のフォークバッファに格納されていたスレッドFが、隣接するプロセッサ部PE3が空き状態になったために実行可能になり、プロセッサ部PE3でスレッドFの実行が開始される。

【0099】その後、図23(h)に示すように、スレ

スレッドCの実行によりスレッドEの実行が確定し、スレッドFの実行の中止が決定される。これにより、プロセッサ部PE3のスレッドFに対してkillコマンドが送信され、プロセッサ部PE3のスレッドFは実行を中止する。以上のようにして、スレッドの投機的実行が行われる。このように、第2の実施形態は、各プロセッサ部PE0～PE $n-1$ がスレッドの管理を分散して行うため、図1に示すスレッドマネージャ1が不要となり、図1よりも装置全体の構成を簡略化できる。また、複数のプログラム経路を同時に投機的に実行してプログラム的高速処理を図る点では、第1の実施形態と同様である。

【0100】なお、第1および第2の実施形態で説明した投機的マルチスレッド処理装置は、ハードウェアで構成しても、ソフトウェアで構成してもよい。例えば、装置全体をワンチップ化すれば、高速処理が可能な高性能の中央演算処理装置(Central Processing Unit)を得ることができる。

【0101】

【発明の効果】以上詳細に説明したように、本発明によれば、複数のプロセッサによりプログラムを投機的に実行する場合に、プログラム内の複数の実行経路を同時に投機的に実行するようにしたため、プログラムを単一の実行経路で実行するよりもプログラムを効率的に処理でき、並列処理によるメリットを十分に生かしたマルチスレッド処理が可能となる。

【図面の簡単な説明】

【図1】投機的マルチスレッド処理装置の第1の実施形態の概略構成を示すブロック図。

【図2】各プロセッサ部の内部構成を示すブロック図。

【図3】プロセッサ部内のレジスタの詳細構成を示す図。

【図4】スレッドマネージャの内部構成を示すブロック図。

【図5】各プロセッサ部内のロードストアバッファの詳細構成を示すブロック図。

【図6】図1の装置のプログラム実行例を示す図。

【図7】図6の時刻 $t_2 \sim t_4$ でのスレッドマネージャの内部状態を示す図。

【図8】図6の時刻 t_2 でのスレッドマネージャの動作を説明する図。

【図9】各プロセッサ部がスレッドマネージャに通知するコマンドを説明する図。

【図10】スレッドマネージャ内の制御部の処理を示すフローチャート。

【図11】fork処理を具体的に説明するフローチャート。

ト。

【図12】term処理を具体的に説明するフローチャート。

【図13】kill処理を具体的に説明するフローチャート。

【図14】commit処理を具体的に説明するフローチャート。

【図15】send処理を具体的に説明するフローチャート。

【図16】notify処理を具体的に説明するフローチャート。

【図17】スレッド間でのレジスタの転送を説明する図。

【図18】スレッド間でのメモリデータの転送を説明する図。

【図19】投機実行状態のロードが発生した場合の例を示す図。

【図20】投機実行状態のストアが発生した場合の例を示す図。

【図21】投機的マルチスレッド処理装置の第3の実施形態の概略構成を示すブロック図。

【図22】6台のプロセッサ部の接続関係を示す図。

【図23】各プロセッサ部の実行状態が変化する様子を示す図。

【図24】あるプログラムにおけるスレッドの実行手順を示した流れ図。

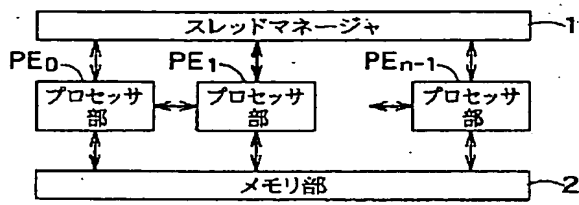
【図25】図24の流れ図を4台のプロセッサ部で並列実行した場合の従来方式によるスレッドの投機的な実行状況を示す図。

【図26】投機的に実行したプログラム経路と実際の実行経路とが一致しなかった例を示す図。

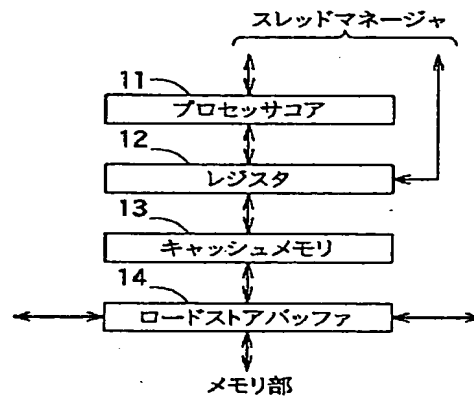
【符号の説明】

- 1 スレッドマネージャ
- 2 メモリ部
 - 11 プロセッサコア
 - 12 レジスタ
 - 13 キャッシュメモリ
 - 14 ロードストアバッファ
- 21 プロセッサ状態表
- 22 スレッド実行表
- 23 スレッド関係表
- 24 コンテキストバッファ
- 25 制御部
- 26 FIFO型バッファ
- 27 レジスタ間転送部

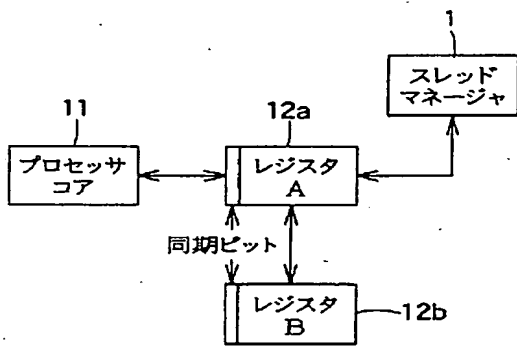
【図1】



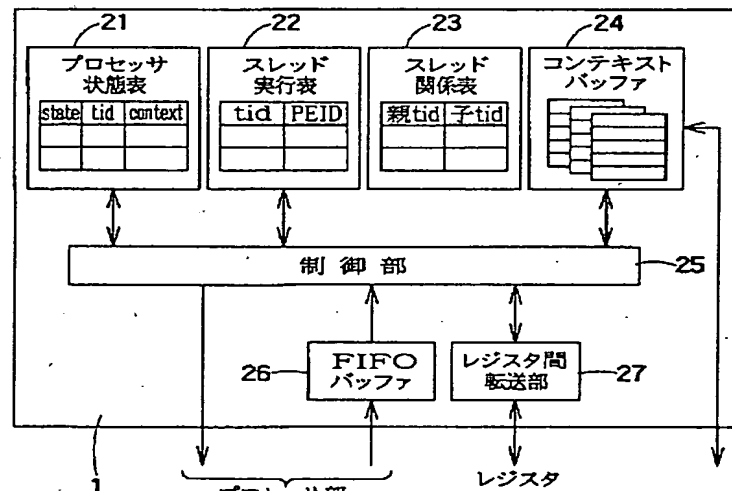
【図2】



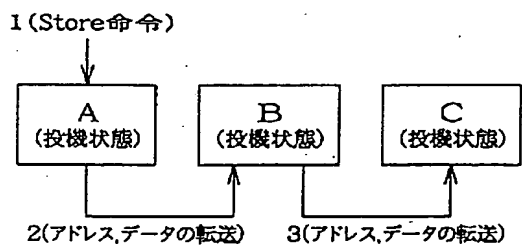
【図3】



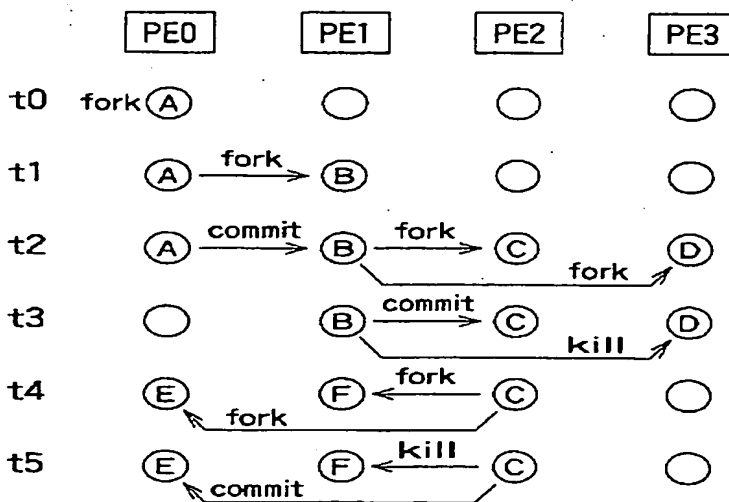
【図4】



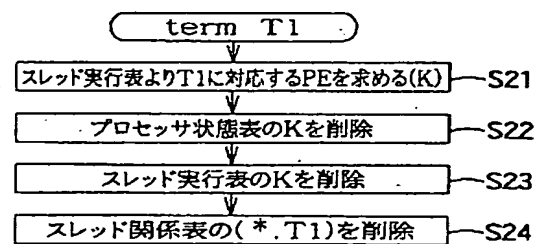
【図20】



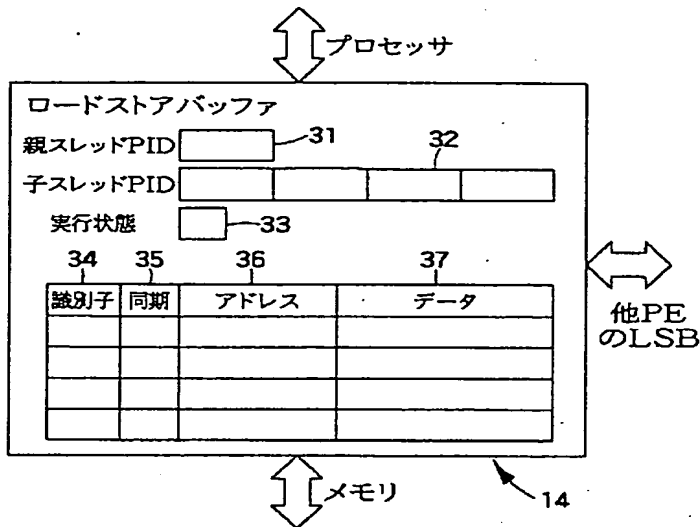
【図6】



【図12】



【図5】



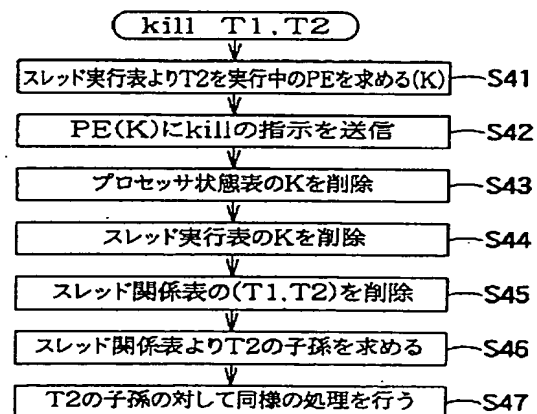
【図9】

コマンド	機能
(1) fork T1, T2 addr	スレッドT1からスレッドT2の起動を要求する
(2) term T1	自己のスレッドの実行終了を通知する
(3) kill T1, T2	スレッドT1からスレッドT2の強制終了を要求する
(4) commit T1, T2	スレッドT1からスレッドT2の投機実行状態から確定実行状態への遷移を通知
(5) send T1, T2, R	スレッドT1からスレッドT2に対してレジスタの内容を送信する
(6) receive T1, T2, R	スレッドT1から転送されたデータをスレッドT2で受信する
(7) check T1, T2	スレッドT1からスレッドT2へのメモリ転送が完了したか否かをチェックする
(8) notify T1, T2 addr	スレッドT1からスレッドT2にメモリのアドレスとデータを送信する

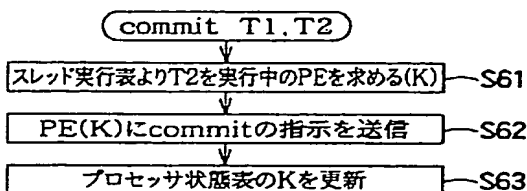
【図7】

t2	プロセッサ状態表			スレッド実行表		スレッド関係表	
	TID	state	context no.	TID	PEID	親TID	子TID
	A	確定	0	A	0	A	B
	B	確定	1	B	1	B	C
t3	C	投機	2	C	2	B	D
	D	投機	3	D	3		
t4							

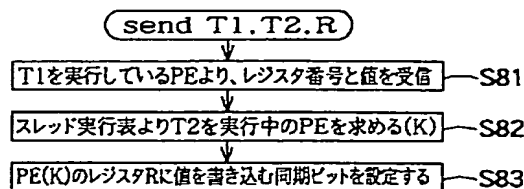
【図13】



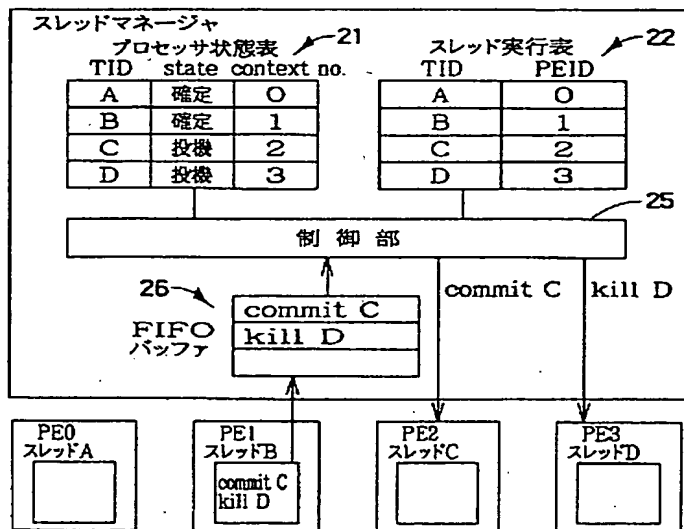
【図14】



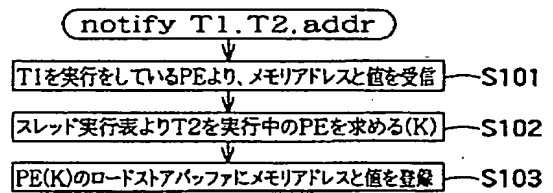
【図15】



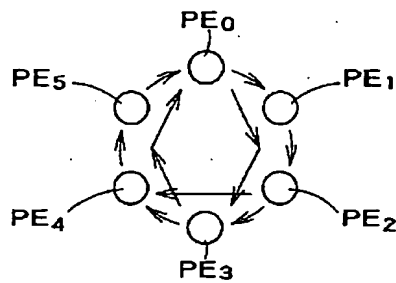
【図 8】



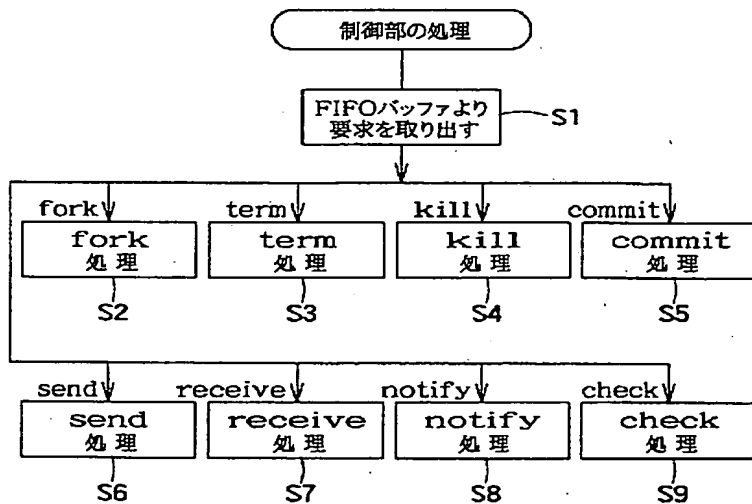
【図 16】



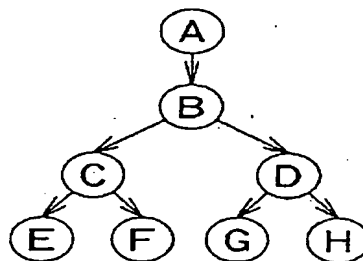
【図 22】



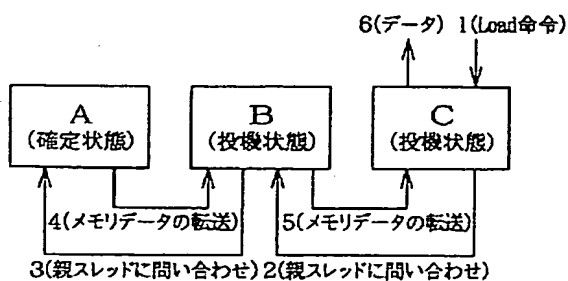
【図 10】



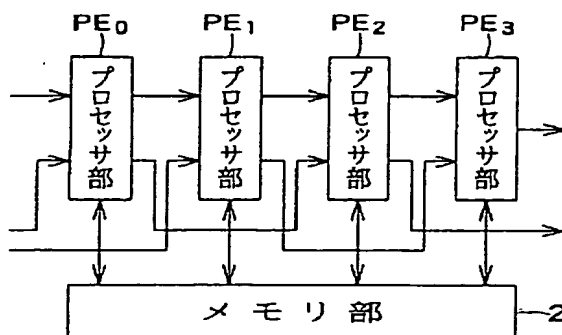
【図 24】



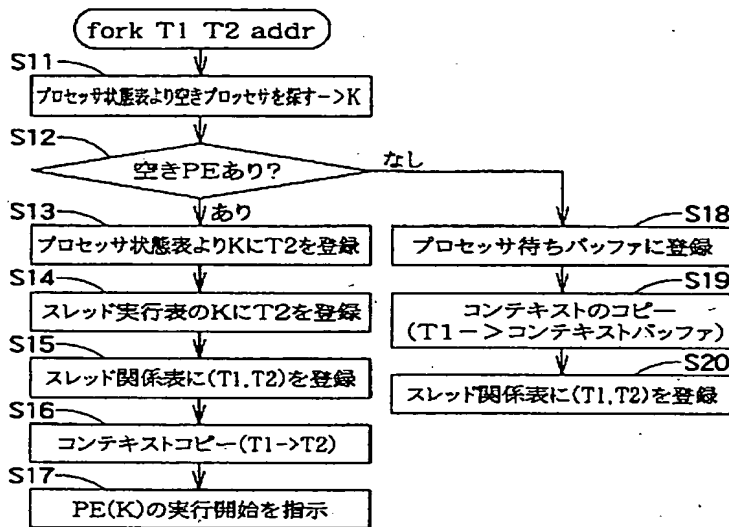
【図 19】



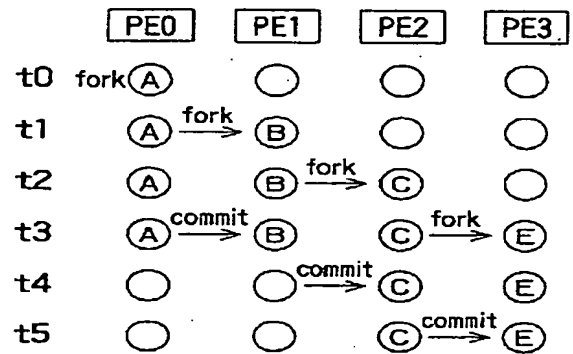
【図 21】



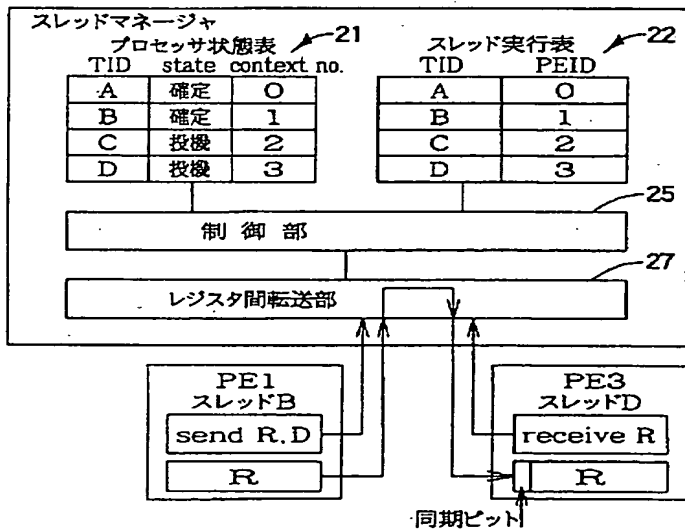
【図11】



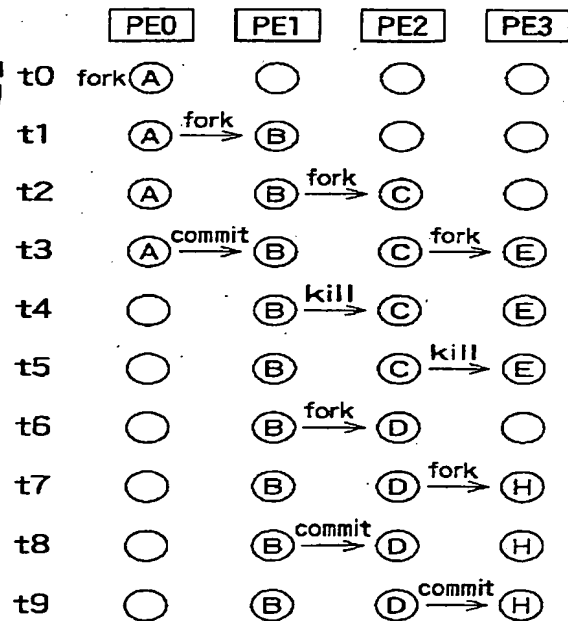
【図25】



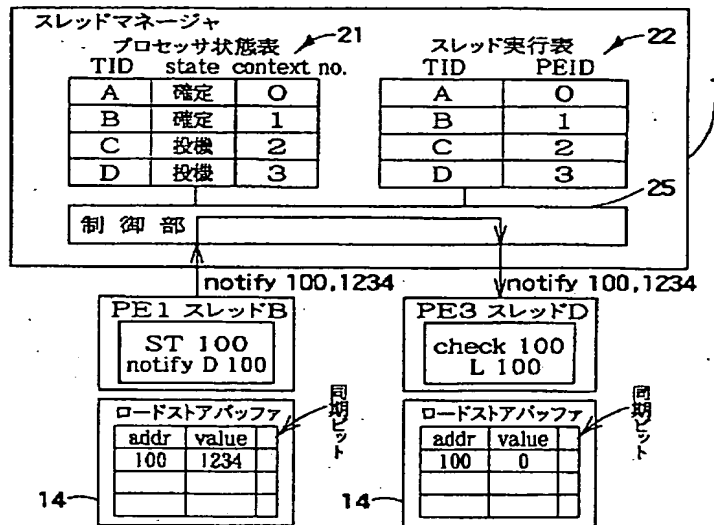
【図17】



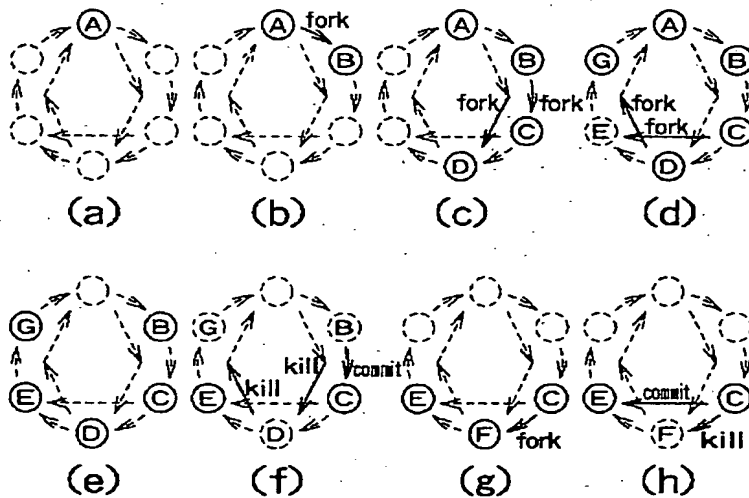
【図26】



【図18】



【図23】



フロントページの続き

Fターム(参考) 5B045 GG11

5B098 AA10 FF07 GA05 GA08 GB09

GC10 GC15 GC16 GC19 JJ09